QBIS Authentication

Integrating an Authentication System Into QBIS

Degree Project Report in Chalmers' Data Program

Jessica Barai

Erik Gjers

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022
www.chalmers.se

Degree Project Report 2022

QBIS Authentication

Integrating an Authentication System Into QBIS

Jessica Barai
Erik Gjers

CHALMERS

Department of Computer Science and Engineering
Division of Data
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

QBIS Authentication
Integrating an Authentication System Into QBIS
Jessica Barai
Erik Gjers

Supervisor: Piyumal Ranawaka, Department of Computer Science and Engineering
Examiner: Jonas Duregård, Department of Computer Science and Engineering

Degree Project Report 2022
Department of Computer Science and Engineering
Division of Data
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: QBIS Logo.

Typeset in LaTeX
Gothenburg, Sweden 2022

Abstract

The process of identifying a user's identity in a system is called authentication. This process requires high security and a user-friendly interface, since the authentication system is often the first section of a service a user encounters. The scope of this project is to implement an independent authentication service for QBIS. QBIS is a project- and time-management product and owned by QLogic AB. A majority of the project consisted of researching what leading authentication systems were available on the market that fulfill QBIS' requirements and selecting one to implement with QBIS' systems. The project's scope also included the implementation of a REST API including automated documentation for QBIS' web services. The open source authentication project IdentityServer4 was selected and implemented. The implemented authentication service offers most functionalities that QBIS offers today. The product was not launched since it needs further development, but much of the structure and design was implemented. A REST API connected to QBIS' web services was implemented alongside automated documentation on its methods. Consumers of the API are able to authenticate themselves with the new authentication system. There were many factors that went into the decision of which authentication service to implement including licensing, pricing, functionalities, maintenance and familiarity. The project in its entirety is the majority of the journey towards an independent authentication and authorization server with expansion of current features to be implemented.

Keywords: Authentication, Authorization, Entity Framework, IdentityServer4, OAuth, OpenID Connect.

Sammanfattning

Nästan alla system använder sig av någon form av autentisering för att identifiera användare. Ett autentiseringssystem ska vara användarvänligt men också ha höga säkerhetskrav på sig. Det finns många authentiseingssytem på marknaden, i det här projektet har en mängd sådana system undersökts. Syftet med det här projektet är att hjälpa företaget QLogic AB att ta fram ett nytt sätt att authenticera sina användare i deras affärs- och redovisningssystem, QBIS. Företaget har krav på ett authenticeringssystem som underlättar en eventuell utökning av deras affärs- och redovisningssystem men som samtidigt bibehåller all funktionalitet som finns i deras nuvarande authenticeringssystem idag. QLogic AB efterfrågar även ett REST API för QBIS tjänster där användare kan identifiera sig via det nya autentiseringssystemet.

Efter undersökningar av open source projekt hittades ett projekt som mötte kraven, IdentityServer4. Ett demo exempel implementerades först för att demonstrera produkten. Sedan återskapades mycket av funktionaliteten som finns i QBIS nuvarande system samt ett REST API där användare kan logga in genom IdentityServer4. Även automatisk dokumentation för API:et implementerades samt ett administrationsverktyg för konfigurationer inuti IdentityServer4.

Samtliga deltagare är nöjda med arbetet som utförts. Tjänsten kommer användas i QBIS testmiljö för att bearbeta projektet vidare.

Nyckelord: Autentisering, Auktorisering, Entity Framework, IdentityServer4, OAuth, OpenID Connect.

Acknowledgements

We are incredibly grateful to Martin Augustsson at QLogic AB for creating and granting us the opportunity to work on this project and his assistance in organizing the project. A special thanks also goes to the head developer at QBIS, Can Peskersoy for the technical help, directives and advice. We would also like to state our appreciation for the entirety of the QBIS staff as they have been superbly welcoming and made QBIS wonderful to be a part of. We would also like to thank Piyumal Ranawaka, our writing advisor at Chalmers, for his advice and direct approach. We would also like to thank Sakib Sistek at Chalmers for his advice, discussion and always lending an ear.

Jessica Barai and Erik Gjers

Contents

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface
CUP	Company, User, Password
EF	Entity Framework
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDP	Identity provider
JSON	JavaScript Object Notation
JWT	JSON web token
OIDC	OpenID Connect
PKCE	Proof Key for Code Exchange
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SQL	Structured Query Language
SSO	Single Sign-On
TLS	Transport Layer Security

1
Introduction

All systems require a secure and user friendly method for identifying users. Authentication is the process of verifying a user's identity. A good authentication system should be accessible and welcoming, since the authentication system is often the first section of a service that a user encounters.

1.1 Background

QBIS is a web-based time- and project management system created and maintained by QLogic for almost 20 years. QLogic plan to update their authentication services for higher security and scalability. QBIS has over 800 customers and over 20000 users. QLogic is expanding their market with a larger international reach while their current main market is in Sweden, which is 98% of their customer base[1].

1.2 Purpose

The purpose of this thesis is to provide a better means of authenticating users for QLogic and integrating such a service into the current QBIS system. The means involve both the ability to authenticate users through an independent authentication and authorization server as well as an API that allows users to authenticate.

1.3 Research Problems

The initial research problem the thesis will attempt to answer is concluding what authentication service is best for QBIS. In order to evaluate the initial question it is essential to understand how to best evaluate the service and how that service would integrate into QBIS' current systems.

1.4 Aim

The project's aim is to integrate a new authentication system into the QBIS product and develop a REST API, an interface for communicating the data in a standardized fashion, to handle the authentication services. The service needs to cover the current authentication features which QBIS' current authentication system offers.

1.5 Limitations

The project's focus is on authentication and authorization, verifying users' identity. The project is also limited to QBIS system and not any other QLogic products. The system that is tested is QBIS test application and implementation on the sold product does not fall under the scope of the project.

1.6 Previous Research

The online environment consists of many protocols that are in use and the service needs are constantly changing as new services fulfill prior ones. While some use SAML such as in [2] where servers are moved from on premises to a cloud base others find other means. It can be seen that RESTful APIs have increased in popularity - something that [3] attributes in part to the change in data formats that are being sent online, with JSON being more popular as an incredibly flexible data format. While all these protocols are the widespread standard of authorization and authentication there is research done to find vulnerabilities in implementations of these such as studies done by [4]. There is little consensus about how much authentication is required for various tasks, but comparisons are made such as those by [5].

2
Theory

The following section include basic terminology and technical aspects allowing an understanding of this project's scope and problems.

2.1 Basic Communication

Much of software development involves delivering data from an external source to a client that intends to utilize it. A client can be a piece of hardware or software that wants access to a service, whether that service be data or functionality. Sending something back from a function, server or device is designated returning. There are a variety of ways this can occur[6].

2.1.1 Polling

Polling is when the source receives requests repeatedly from the client in order to complete the transaction. These requests can also be something as simple as a single byte read, but the same principles still apply for those situations as with larger data payloads. For each request the source returns a message indicating the status of the request, whether that be the request is invalid, it is being initialized, it has not completed yet or that it returns the requested data.

Polling is generally inefficient and is to be avoided. It is not an expensive process to send and receive a few messages, but this form of communication does not scale well as you may have a source that is receiving several hundred requests at any given moment. If all of those clients are constantly requesting the server it may overwhelm the communication channel. It is also suboptimal if the client would like the requested information as soon as possible, as that means it would require a high polling-rate, meaning it would send many requests over a short time period, in order to shorten the delay[7].

2.1.2 Interrupts and Events

Interrupts and events are possible solutions to the problems of polling. Interrupts are signals that inform the hardware that it needs attention, interrupting the current flow and executing handling. Events are the software equivalent of an interrupt. Regardless of whether an event or interrupt is used they both trigger the handling of the delivered data as soon as possible. This means the smallest possible wait period, since the client will receive a response as soon as the request has been

handled, and amount of wasted resources, as only one request needs to be sent and only one response[7].

2.1.3 Interfaces

An interface is a programming term for a description of how communication can occur with an object. An example would be play and pause. Play and pause exist in many different media: In remote controls and voice recorders. Regardless of the media the interface remains the same.

Programming interfaces allow us to swap out implementations of functions. Once the interface is defined and documented it means that others can use the same protocol to allow them to be swapped[8].

2.2 Web Communication

All web communication occurs between a client, a software that is accessing the web, and a server, a service provider that relays information to some specific addresses on the web. These entities can be referred to as being client-side or server-side. When transferring data it is imperative to understand how this communication occurs and when it is secure from external view[9].

2.2.1 Hypertext Transfer Protocol

Hypertext Transfer Protocol, or simply HTTP, is a protocol that is used for communicating data between a user's computer and a website hosting on a web server. A HTTP call or request consists of a request method, sometimes called a HTTP verb. The most common types of methods and potentially most important are GET and POST.

GET retrieves information for the user. POST attempts to change the state of the server they are communicating with. The information that the user requests is contained in the body of the HTTP response. In the case of a POST the HTTP request body contains the information to be posted. Beyond a body HTTP request and responses also contain metadata, data about itself, in the HTTP header. In the HTTP header there is information such as where the request came from, or what type of file is in the body[9].

2.2.2 HTTPS

HTTPS stands for Hypertext Transfer Protocol Secure and is HTTP with added security. The security added is called Transport Layer Security, or TLS. When a user attempts to connect via HTTPS, the site's security certificate is shared. In this certificate is a public key that the user can use to encode a message to get a session key from the server. The initial message is decrypted by using the private key held secret by the server. The session key is then used to encrypt the message for the remainder of that connection. All communication is thereby unintelligible to outsiders[9].

2.2.3 HTTP status codes

A HTTP status code is a server response to a client's HTTP request. It is useful for identifying errors in the requests to the server, usually along with an error message with details about the request[10].

2.2.4 Application Programming Interfaces

An application Programming Interface, or API, is a specified communication for software. An API defines, through documentation and programming, how it is communicated to and from allowing a program to communicate to it to utilize the information or services it provides. The communication between protocol differs depending on which type of API there is[11].

2.2.4.1 Over-Posting and Mass Assignment

One aspect of API design is to make sure that the consumer of an API should not be able to write to properties in the database that should not be manually controlled by the user, so called over-posting or mass assignment. The consumer should not be able to view or change sensitive properties in the database, such as password or personal identity number[12]. The solution to these problems is protecting the data by having the methods not write directly to the sensitive data, but instead to a container that is then filtered or reconfigured.

2.2.4.2 Endpoints

An API may not and most often will not have a single point of contact. Instead an API may have multiple points of contact offering different services. These endpoints have different addresses allowing them to be differentiated[13].

2.2.4.3 Simple Object Access Protocol

Simple Object Access Protocol, or SOAP, is a protocol defining how an API should communicate. SOAP is popular in enterprise and was a standard, but has lost popularity. SOAP has a single endpoint to perform all of its communication[14].

2.2.4.4 Representational State Transfer

Representational State Transfer, or REST, is a set of guiding principles for API communication. It is not a strict protocol and instead a set of guidelines. These guidelines include things such as a uniform set of commands to contact the API, streamlined interactions due to data saving and being stateless. Stateless means that each request is independent and unconnected to other requests. APIs built with these principles in mind are called RESTful and named REST APIs[15].

2.2.5 JavaScript Object Notation

JavaScript Object Notation, commonly and hereinafter JSON, is a text format to indicate the properties of a given data class. It is one of the most common ways of sending information between applications[16]. JSON is a list of properties with given values. These properties can then be read as key-value pairs, where a given key corresponds to a certain variable or value. A key can correspond to a set of values. Figure 2.1 shows an example JSON string.

```
{
        "book": {
                "title": "Example book",
                "author": "Erik Gjers",
                "chapters": [
                        {"title": "Chapter One", "text": "Once upon a time..."},
                        {"title": "Final Chapter", "text":"... The End."}
                ]
        }
}
```

Figure 2.1: A JSON string. This string is an example of how a book might be formatted. Note that the same property "title" appears both within chapters and the book itself.

JSON can be read and generated in all modern programming languages and is therefore useful in contexts where you may have entities that are using different languages and need to communicate[17].

2.2.6 Tokens

Tokens are encrypted pieces of information that act as digital keys. They allow access to a resource. There are two main kinds of tokens discussed in this report: access tokens and identity tokens. Both of these contain the information allowing an application or user access to a resource. These tokens can then be transmitted online[17].

2.2.6.1 JSON Web Token

A JSON web token, commonly and hereinafter JWT, is a form of token using the JSON format. A JWT consists of three parts: header, payload and signature. In figure 2.2 an example token is displayed.

Figure 2.2: An example JSON Web Token. It consists of three parts separated by dots. The part in red is the header while the part in green is the payload. These two parts are legible as they are simply encoded. The part in blue is the signature which is generated by a variety of methods. It uses the header and payload alongside a secret piece of information so that the token cannot be modified without having to alter to signature.

The header and payload are encoded so that anyone can read them. The header contains information on what kind of token it is and what algorithm is used to encrypt the signature. The payload has information known as claims that are key-value pairs as they are following the JSON format. There are some standard claims such as "sub" that indicates who the holder of the token is or "iss" for which application created the token. The final part of a JWT is the signature. The signature is the entire contents of the token encrypted with the hashing algorithm listed in the header alongside a secret that the issuer of the token knows. This way the issuer can verify if the token has been tampered with or not. JWTs can be also be unsigned meaning that the signature is empty and hashing algorithm is listed as none. These are more useful in contexts where verification of the source is not necessary[17].

2.3 Authorization

Authorization is the process of granting permissions to a user or application. These permissions can be such things as writing or reading from a source of information or any number of abilities[18].

2.3.1 OAuth 2.0

OAuth 2.0 is a protocol that deals with authorization on the web. Companies such as Google, Facebook and Spotify use this protocol. It provides a variety of methods to authorize a client, a software that a user is applying, to various resources. A verified user can authorize their software to access the resources that the user has permissions to access.

In OAuth there is generally a separate authorization service or server, resource owner and user-client. If a user attempts to connect to a protected resource and is not verified, the user-client will be redirected to the authorization service. Once the user is identified, they can then authorize the client to access the resource if the user has that permission. The client is then granted an access token to indicate

to the resource that the client has authorization. OAuth only deals with authorization, but has an additional protocol on top of it to handle authentication[18].

2.4 Authentication

Authentication is a process whereby a user is able to to be verified or prove their identity. Authentication can then be used to allow users to gain access to sensitive user data or services that require an identity by authorizing the clients that they are using. When a user provides their credentials, commonly a username and password, to an application, an authentication service must handle the request and inform the application, or separate authorization entity, if the user is who they are intended to be[18].

2.4.1 Credentials

The credentials of the user may not be username and password. There are many forms of credentials to verify the identity of a user. Credentials can be in the forms of a code that the user has received or even a specific item that the user has, that can be identified in some way via a unique code[13].

2.4.2 Means of Authentication

There are many ways a user can prove their identity. Below is a non-exhaustive list of options that are relevant to the project at hand.

2.4.2.1 Basic Authentication

Basic authentication is the simplest form of authentication in use. When a user logs in, a header is sent with password and username arguments to the back-end service. For every request made to the server, the username and password are also sent in order to authenticate the user. This process exposes the credentials often, making it more vulnerable to outside access. If this information was sent over a network that was not secure someone else could access the credentials thereby allowing them to authenticate falsely[13].

2.4.2.2 BankID

BankID is an electronic identification system. BankID utilizes personal identity numbers which are unique 10 digit codes tied to a Scandinavian citizen. This number contains the date of birth and gender in most cases[19]. By using Swedish personal identity number and a personal code in a connected app, users can authenticate themselves. The method for acquiring authentication from BankID involves polling when the client application sends repeated requests to the BankID API. On a successful authentication the API returns a JWT with user specific information including their personal identity number.

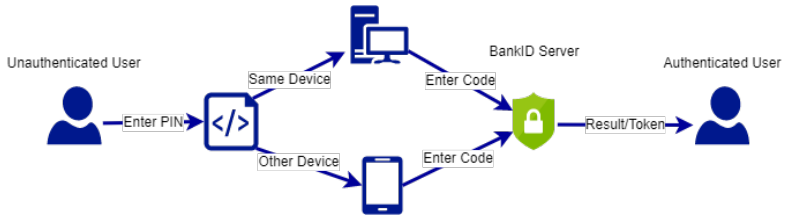

Figure 2.3: The workflow for a user using BankID. Users can select to authenticate on the same device or a different one. The user has to verify with a code regardless of method chosen.

In figure 2.3 the flow for a user authenticating by BankID can be seen. The user enters their personal identity number in order to use the service. They can then choose to authenticate themselves on the current device or a separate one. In order to gain access to the service the user must be registered with a bank for the service. Once ordered it is attached to a file on the device that is desired and a required code is set for its use. This code is required for each use of the BankID.

One of the rules limiting the implementation of BankID is ID-exchanging. ID-exchanging is when a user tries to create a new identity through their current one. This could mean that a user authenticated with their BankID unlocks additional permissions even when using username and password to authenticate themselves in the system in the future. This is strictly prohibited by the service[20].

2.4.3 Sessions

One type of authentication is session-based authentication. The most traditional is cookie-based server-side sessions, where cookies are small client-side files attached to a browser containing user information. The process starts with users filling in credentials and submitting these to an application. The application validates the credentials server-side, which then stores and returns a session ID to the client-side. The ID is saved as a cookie in the browser. Every request coming from the client is now sending the session ID to the server which crosschecks it with the stored ID. This means the credentials required for authentication are only sent once[13].

2.4.4 Token-based Authentication

Token-based authentication starts with the user sending credentials to the application and behaves in the same manner as session-based authentication. The server returns an encoded token instead of a session ID. Tokens can be decrypted by the server to check their validity. The user sends their token for actions requiring authentication. The difference from session-based is subtle, but this requires no storage server-side. This means a token based solution is generally more scalable than session based systems[13].

2.4.5 OpenID Connect

OpenID Connect, or simply OIDC, is an additional layer that is used in the OAuth 2.0 protocol. This layer allows clients to authenticate an end user. Once authenti-

cated the user can authorize various clients to access protected information[21].

2.4.5.1 Authentication flow

OIDC and OAuth define a variety of authentication flows, also called grant types. One of the most common types of flows for web application projects is the authorization code grant type. The flow consists of two processes for the user's client: One with the authorization server and one with the resource owner. The user logs in at the authorization server, potentially having been redirected from the resource owner, if the resource owner and request is valid. The authorization server confirms the authenticity of the user with information inside its data store. Once the user is successfully logged in the server sends an authorization code to the client. The client sends this code to the resource owner which then contacts the authorization server and returns an access token[13].

2.4.5.2 PKCE extension for extra security

When a client is making a request to get the authorization code through the browser it is possible for a third party to acquire that code. There are ways of preventing the code from being useful to a third party by means of a security extension. The extension is called Proof Key for Code Exchange, or PKCE. When enabling this extension the client must generate a random value and encrypt it, returning it to the user. The client is the only one who knows the value this since it generated it. The user is then authenticated in order to receive the authorization code. When using the authorization code to gain a token the encrypted value must also be passed to the resource owner. The resource owner then passes the encrypted value and the actual value to the authorization server and this acts as an additional layer of security, ensuring that the request in the later stage is related to the request for the code in the earlier stage.[13].

2.5 Databases

A database is an organized collection of structured data or information stored in a computer system. The database is usually controlled by a database management system, known as DBMS, a software system acting as a communication channel between the user and the database storage[22].

2.5.1 Structured Query Language

Structured Query Language, commonly referred to as SQL, is the language used in relational databases. The language is defined by an ISO standard, [23]. There are many implementations of the standard such as PostgreSQL, MySQL and Microsoft SQL Server to name a few. Relational databases have tables that hold their data which then relate to each other. A table has columns and rows. A row is an instance of data while a column is a trait of that data. While other forms of databases are gaining popularity, relational databases are the strong majority of all databases. A

request for data is known as a query. A query will have a variety of constraints applied such as which tables to fetch data from, how those tables should be merged, which columns should be fetched and what values those columns should have[22].

2.6 Authorization and Authentication Servers

Authorization and authentication can be integrated directly into an application. It can also be an independent service that has its own server. In such cases applications that require authentication and authorization redirect to this server and it will redirect the user back when appropriate. This section covers some of the concepts regarding such independent authorization and authentication services[13].

2.6.1 Identity Providers

An identity provider, IDP hereinafter, authenticate users, provide identities to users and validate those identities. There are many ways for an IDP to validate a user, but the most common way is by knowledge. If the user knows their password the IDP will return an identity that the user can use with the appropriate permissions to use the service they are requesting[13].

Most modern applications need some sort of security check at multiple points in the system. In figure 2.4 an example of the flow of an identification is shown. A user needs to authenticate themselves from their browser to a web application and this, in turn, needs to be authorized to be able to get data from a web API or other service provider.

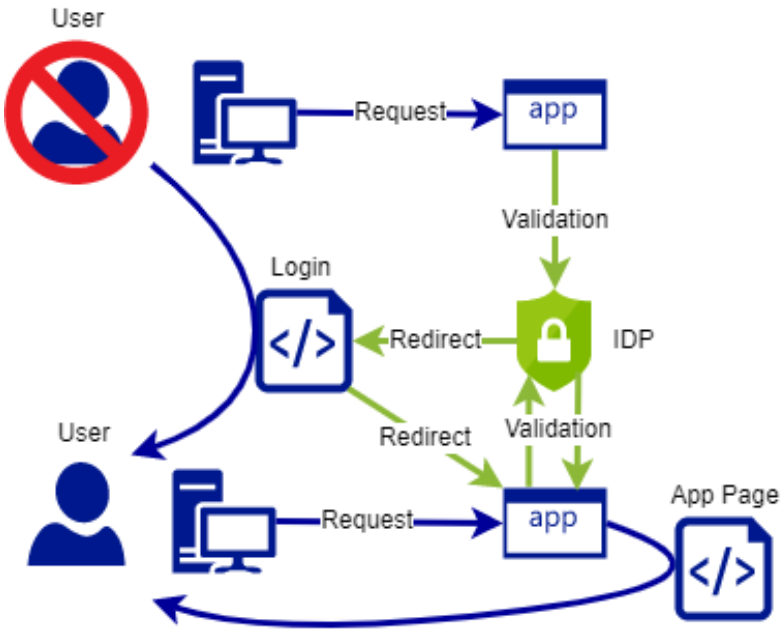

Figure 2.4: How Authentication works in most modern applications. The user must be authenticated and then authorize the application in order to access the application page. If the user is not authenticated a redirect will occur allowing the user to authenticate themselves[13].

The reason for such a flow is centralizing the authentication and authorization functionalities. An example without centralization would be a security system where the authentication is being integrated into the system, without the usage of an independent IDP. The application would require its own login page and cookie to be issued. If several more applications were created then they would each require their own login page, cookies and implementation. It is far more scalable and user-friendly to decouple the authentication and authorization services by centralising those services into an IDP. This allows independent updates and management of the authentication system so that such changes do not affect other products associated with the IDP and need to be implemented anew there.

To centralize the authentication part of an application means that the authentication server will know who your users are, authenticate them and generate a token. These services mean that the applications that utilize the IDP do not need to handle information such as credentials which is helpful in limiting data breaches.

2.6.1.1 Security Assertion Markup Language

Security Assertion Markup Language, commonly and hereinafter SAML, is a standard for sending and receiving identities. What SAML allows sending of identities to other trusted service providers securely. This allows for single sign-on, SSO, which is a common feature in many web applications. SSO entails that a user that is logged in through one service can be logged into another, without being asked for credentials. SSO is implemented by using SAML to transfer the identity from one service provider to another[24].

2.6.1.2 Shibboleth Service Provider

Shibboleth is an open-source software implementing a SAML protocol. In QBIS Shibboleth Service Provider is used to connect with external IDPs[25].

2.7 .NET

.NET is an open-source development framework created by Microsoft. The framework offers tools for web APIs and mobile development. The .NET framework mainly runs on Windows' operating system while .NET core, the newer version, is a cross-platform framework. ASP.NET core is a part of the .NET framework and is used for developing web applications[26].

2.7.1 Entity Framework

Entity Framework, or simply EF, is a object-relational mapping framework working as a component in the .NET framework. In data-base first implementations developers can abstract objects from a database and its tables by using EF. This allows the the programmer to simply deal with objects rather than querying a table in the database for information. EF connects to the database and creates a session in

the form of a DbContext, short for database context. This context can be used to perform queries and save information to the database[26].

2.8 IdentityServer4

IdentityServer4 is the name of an authentication and authorization server software that implements OIDC and OAuth 2.0. The software is a framework in ASP.NET core and needs to be configured for the system it will be used in. IdentityServer4 acts as a middleware that connects the framework for users to the framework for the authorization to resources. It handles the authentication of the user, creation of endpoints and protection of resources.

The tokens generated by IdentityServer4 are JWTs. Such a token contains information about the user and is signed by the authentication service. IdentityServer4 acts as a centralized token service, both for creation and validation for the applications. Its job is to authenticate the users and authorize appropriate clients for those users[13].

2.8.1 Claims

IdentityServer4 emits data about clients and users into tokens using the claims in the payload of the JWT. Claims can be customised for users or applications and can be added or removed as such.[13].

2.8.2 Scopes

Scopes are a license for clients to gain access to various resources and functionalities. For example, a scope could specify if a client has read and/or write access to a web API. This is handled and configured for each client in the IdentityServer granting them a set of scopes to utilize. Scope is generally a claim inside of token to indicate the permissions the token can grant[13].

2.8.3 Skoruba

In order to configure resources such as scopes and client credentials it is useful to have software that allows those to be changed as needed, without having to write directly to the database from code. Jan Skoruba,[27], wrote an administrator application for handling these sort of scenarios. It has its own authentication service and therefore protects the internal information and functionalities while allowing authenticated users to edit, add or delete roles, scopes, clients and more.

2.9 Tools

In the following sections a list of various tools used for the project can be found.

2.9.1 Agile Software Development

Agile Software Development, or simply Agile, is a collection of frameworks and practices that teams can use to develop software products [28]. One of the foremost ideas of Agile is having a variable scope of the project at hand. This means that what quality and size product becomes developed is not known from the outset of a project, but rather decided based on the amount of time and work input into the project.

2.9.2 Visual Studio

Visual studio is an integrated development environment, or IDE, developed by Microsoft. It mainly focuses on .NET applications which have a variety of versions[29].

2.9.3 Postman

Postman is an application that allows formulation of HTTP requests. It allows you to set parameters and turn them on or off at will, making testing various requests much simpler than simply writing them out[30].

2.9.4 Swagger Documentation

Swagger is a tool used for documentation, design and ease implementation of RESTful APIs[31].

2.9.5 Git

Git is the dominant version control system on the market. Version control software allows developers to easily share code and work on code simultaneously. The code is then merged together at a later point. Generally a programmer will work in their own branch, or copy of the code, that is later merged into the master or main branch. [32] When writing code with Git, it is done locally. This is then pushed to the remote repository for the code. Other coders can then fetch the changes to the code when they want to update their local copies. The remote repository can be open to the public. If allowed a new set of programmers can take such a repository and fork it. This entails splitting the project into two different branches that are functionally different and could or could not be merged back together.

2.9.6 Atlassian Products

Jira is a web application that is used for project planning. This was an additional channel of communication as the entire team could see what each individual was working on. Confluence is another web application that is used for creating a forum of documentation that belongs to the Atlassian suite of products. Belonging to this quite is also Bitbucket which is a version controller that uses Git [28].

2.9.7 Microsoft Teams

For communication with QLogic Microsoft Teams, or Teams, was used.

3
Method

In this section the systematic approach to reach the solutions to the thesis' problems and answer research questions is described.

3.1 Overview

A majority of the project was researching the general concepts within authentication. The next step was to select a product to implement into QBIS which was split into understanding the current QBIS-system and researching products that could fulfill QBIS' needs.

After that had been completed the next step was to implement the product into the system. The project was built in a ASP.NET Core 3.1 environment. The specifics of the working methodology can be found in the sections below.

3.2 Use of Agile Practices

In the project several agile practices were applied. Sprint planning was used where the sprints were 1 week long. Sprints are a a period of time where one designates the tasks that are to be completed in that period. After each sprint a retrospective was performed consisting of reflection and evaluation on whether the goals had been reached and what practices should remain, be altered and/or be removed.

In this project Jira was used for planning sprints and tasks as QBIS uses several Atlassian products for organisation. In this project Confluence was used for documenting meetings and what has been done in the project while BitBucket was used in this project as the primary tool for version control.

3.3 Choice of Authentication Service

The first choice of the project was to select a means to implement the authorization service whether this was by middleware, an out-of-box product or building from the ground up. In this section various aspects of this choice are revealed. Not all choices that were reviewed are listed, but rather a selection of some of the more appropriate services for the project.

3.3.1 Requirements

While looking at the services some requirements were checked that were used for evaluation of various implementation methods.

3.3.1.1 Maintenance

The requirement for maintenance means that the authentication system needs to be scalable, preferably maintained and supported by some reputable and reliable source. It should also be understandable and function within the environment of QBIS products meaning that they don't need to hire further expertise to support it.

3.3.1.2 Pricing

Lower is better in this case and free is the best alternative. This led to some open-source projects being the primary potential choices considering some of them are free of charge.

3.3.1.3 Licensing

An appropriate license, a document that provides rights to users for the product, is vital as QLogic is a for-profit company. Many of these services offer free licenses to smaller companies. Since QLogic does not tend to fall into such a category generally a subscription becomes necessary. Freeware licenses that require the software only be used in other freeware would also be off-limits such as freeware that is licensed with a copyleft license[33].

3.3.1.4 External authentication system

One of the most important requirements for the authentication system to be implemented is the independence of such a system. QBIS preferred an authentication server which should work as an independent unit in the system. If the authentication system is integrated into the current QBIS system then it may suffer some of the effects of the current system. If QBIS were taken down momentarily for updates or other reasons then all sessions stored in memory would be cleared. This would cause users to have to log in again. By centralizing the authentication system all connected systems will be more maintainable since authentication and authorization is decoupled from the working of the products. The functionality can also be changed to allow user logins to be stored in an operational database and/or tokens.

3.3.2 Initial review of services

Services were found by search, using key terms such as authentication as a service, AaaS, authentication and authorization. This was helped by looking for selections of such services with keywords such as top or best. A list was then made of various services and limited down to thirteen choices by means of reviewing information if the service was an independent authentication and authorization server. Services were often independent implementations of authentication methods, which did not

qualify. From these thirteen the requirements were reviewed for pricing, licensing and, to a lesser extent, functionality. These options were then discussed with QLogic and 5 options were selected to investigate further, listed below.

3.3.3 Out-of-box

Out-of-box, or preconfigured, solutions were looked at as potential candidates. There were major issues with out-of-box systems. QBIS does not run a standard setup for authentication, being multi-tenant and using the CUP format. While an out-of-box solution could be configured for this case such options are realistically improbable.

3.3.3.1 Keycloak

Keycloak was an interesting alternative for authentication services. Most of the documentation assumed a code-first approach. It is simple to get going and use it for minor projects. Connecting it to QBIS' current database would be a challenge and would require a middleware to abstract the multitenancy aspect of the database[34].

There were several attractive aspects of the software. It is open-source, free of cost and backed by Red Hat, a major contributor to the free software movement and leading within open-source enterprise software.

It is run in a Java environment which is not a detriment in and of itself, but negative when combined with QBIS. Since QBIS runs within ASP.NET and adjacent environments it is not ideal for the developers of QBIS to maintain the server being run within a Java environment.

3.3.4 IDPs as a Framework

Some software does not run as an application that can be configured, but rather provides a set of tools that can assist coding a possible solution.

3.3.4.1 IdentityServer4

IdentityServer4 was the chosen product. It is a middleware and provides strong customizability. Since it is so customizable and in wide use there is plenty of documentation and projects available online. Some of these show solutions similar to those needed by QBIS.

It is widely used and even mentioned many times in .NET documentation[35] when stating options for dealing with identity, authentication and authorization software. The pricing is also ideal as it is free of charge and will be free to use indefinitely.

It is open-source freeware and will reach its end of life cycle in November 2022[13]. This means that it will no longer be receiving updates after this point. This means that this product is weaker in terms of the maintenance aspect as QBIS would need to fork it and perform security patches to use it in the not too distant future.

3.3.4.2 Duende IdentityServer

The successor of IdentityServer4 is Duende IdentityServer. The developers moved away from their free model to a paid subscription model with the option of a community edition which can be used for charities, smaller projects and small companies[32].

Duende shares much with its predecessor, being a framework that works very well in the .NET Core environment. It is also listed in[36] that the identity and login handling packages of the .NET Core environment, was designed to work with Duende. This package is known as Identity.

QBIS has little interest in working with Identity as QBIS uses a legacy database which was not created with Identity in mind. While it has lower costs compared to other alternatives on the market, the functionality here is very close to IdentityServer4's own. QBIS was more interested in seeing the improvements provided by an authorization server product and then making a more informed decision later, on what to dedicate themselves to.

3.3.4.3 FusionAuth

Another IDP of interest was FusionAuth. FusionAuth is also an incredibly flexible tool that can be used as a framework and/or preconfigured. The main strength over IdentityServer4 and the like is that it has a built in solution for multitenancy where by default it runs with a single tenant, making it very easy to add other tenants. This does not solve QBIS multitenancy problem entirely, because it requires a certain database structure. The framework is flexible and a solution could likely be implemented within it[37].

One of the major issues is that the cost would be about six times as much as Duende IdentityServer for QBIS purposes, making it a much less interesting alternative and the reason it was not pursued further.

3.3.5 Creating an IDP

While implementing the OAuth and OpenID Connect protocol to create an IDP could be a lucrative product to develop, the approach has many flaws. In terms of maintainability this is a poor alternative as it would provide a whole product that requires updates regularly. If QLogic wanted to expand their product line to include an IDP product then that would be appropriate, but that was not the purpose of the project at the outset. Developing one's own solution implementing OAuth and OIDC would take a significant amount of time upfront and perpetually as security is a full-time endeavour. This option would only have been used if no other option was available which was not the case. Considering the scope of such a project it was deemed not worth the cost of the investment.

4

System

QBIS current implementation has been developed over more than 15 years. There are a multitude of frameworks applied and far more developers that have worked on the product. The system is intricate and the entire product consists of over 100 .NET projects using a large variety of framework versions. This chapter detail some of these systems that are relevant to the research questions and give an idea of how QBIS' current authentication system operates before any implementations in this project.

4.1 QBIS' Current Authentication System

QBIS' system provides users with a variety of ways to authenticate. By the end of this project all authentication methods should still be available in the new authentication system.

4.1.1 CUP format

Unlike many other credentials, QBIS uses 3 parameters: Company name, username and password. This is called the CUP format within their documentation and hereinafter.

4.1.2 Basic Authentication

Users of QBIS' system can enter the system by specifying their company name, username and password. This type of authentication is a basic https authentication.

4.1.3 Single-sign-on

Single-sign-on, or SSO, is a token-based authentication method in QBIS' system using the SAML protocol for exchanging authentication data between different systems. This is accomplished via Shibboleth. This means that users do not need additional credentials to log into QBIS since they can authenticate themselves via a different IDP connected by Shibboleth instead.

QBIS also provides a SAML endpoint to immediately redirect a user beyond the login page if the user is authenticated to an SSO service. If they do not connect to this endpoint they are instead authenticated when attempting to login on the login page.

4.1.4 Authentication with BankID

To be able to use BankID in QBIS' system users must register their account for authentication with BankID. Once BankID is activated for an account the user is not able to login via basic authentication. This is a precaution added due to ID-exchanging. If a user registered with BankID tries to login with basic authentication in QBIS, the user will receive an error message encouraging them to login with BankID.

QBIS provides their users with two methods of BankID authentication: on the same device or on a different device. If users have their BankID app installed on a different device from where they attempt to log in to QBIS they need to start their BankID app manually and scan a QR-code that the BankID server provides. When users select to authenticate via the same device they are on, they will receive a notification to launch BankID.

On the QBIS login page users are able to select an option to login with BankID. If this option is pressed the QBIS Web API will send a request to the BankID server after acquiring the personal identity number of the user. Since QBIS requires the company name, a separate database table is used here storing company identities against personal identity numbers. This table allows QBIS to identify which company or companies the person is associated with.

If the users are registered in QBIS' system with more than one company they will be redirected to an intermediate page where they choose which company they want to log into. If a user logs in successfully the BankID server returns a valid token along with the user's personal identity number to the QBIS server. This is then used to log the user in normally.

4.2 Database structure

The project can be considered data-first, this means that there is already an established database that is in use. There are varying tables that are of use in the authentication process and others that are useful as information for users to fetch using a potential RESTful API.

The database is of a multitenant structure. Multitenancy is when you have several tenants, or multiple structurally similar databases within the same database environment. For each customer, which is a company, a separate database is held that has the same schema as the other customers' databases. Each contains its own information that should only be accessible by the employees, or users, of that company. In each of these tenant databases is user authentication information. This means that two users can have the same username and password as long as they are at different companies. Each company id and name must be unique.

There is a central database that contains information about how to connect to each of these databases. This database also has additional tables, such as the additional table mentioned for BankID that identifies users' companies based on their personal identity number.

5
System Implementation

The project also included the implementation of IdentityServer4, a REST API as well as an administrator application for editing future clients, scopes and other configuration information.

5.1 Integrating IdentityServer4

The integration of IdentityServer4 into QBIS' systems consisted of several sections and steps. Initially developed was a test environment for IDS4 before moving on to implement a login page, validation and endpoints.

5.1.1 Initial Setup

IdentityServer4 provides a quick-start template that was used for the project. An additional text field was added for the company name. In order to connect appropriately to the database several classes within IdentityServer4 needed to be rewritten.

Clients were set in the startup of the server when a seed command was given. This was altered later. A local database was used for the configuration- and operational-store. In here any information regarding active tokens or similar could be stored and information on clients could be set. Initial tests were done in-memory, but eventually a full database approach was adopted even in the testing stages.

5.1.2 Creating a User Store

QBIS' systems made no use of Identity since it would require a specific database structure to function. In order to utilize some of the tools of IdentityServer4 a customised user storage class was created and used in place of the original. This means that users could be defined from the QBIS' database design and that the new class could be used instead, abstracting away the database users at this layer. In order to get any information on a user the Employee class was made. This class can be populated from the relevant DbContext, from EF, thereby allowing reading and writing of information from the database. This information could then be used to validate any user inputting credentials in the CUP format.

5.1.3 Multitenancy

In order to deal with the multitenancy the user stores were generated from a factory class. The factory generates the user store requested by utilizing the connection

string found in the central database context: The company id determines which user information to fetch and the user information populates the user store.

5.1.4 Endpoints

The endpoints were mostly functional with the initial settings. The classes that needed to be rewritten were related to the generation of tokens and authentication. Rewriting the token generation class allowed the addition and removal of claims. Authentication could not be done with the standard class since it was not using the CUP format and other rules that the validation of QBIS required, such as an active company and user.

5.1.5 External Authentication

In order to demonstrate how external authenticators could be added, BankID was implemented. Active Login's packages for BankID were used for this where most of the functionality and API calls to the BankID API were handled[38].This uses a different section of the QBIS central database as it was initially made to handle multitenancy only by finding the company name, but a table for finding companies by personal identity number of those who use BankID was added to support BankID functionality when it was implemented for QBIS' systems. The Callback method, called after the authentication completes, had logic added to it to handle the selection of the correct user in the multitenant database and, in the case of a personal identity number being attached to several companies, a redirection to a company selection page before finalizing the login of the user.

5.2 Restful API with Entity Framework

As a part for the project a Restful API has been created. This API allows users to fetch and change information that would ordinarily only be altered through QBIS.

5.2.1 Prevent Over-posting with Data Transfer Objects

One problem with implementing a REST API is the risk of a user over-posting, writing to properties in the database that violate security. To prevent this Data Transfer Objects, DTOs, were used. When the consumer of the API uses create or update methods in the API they are, instead of writing to the database directly, writing to a so called DTO class. This class contains fields that a user is allowed to manually change or update, not all fields in the database are accessible to write to for the consumer of the API. The transfer from DTO classes to the actual database is a process occurring internally within the code of the methods. The DTO classes act as an additional protective layer around the database. A DTO class was also the solution to prevent users from reading sensitive information from the database such as the password field. By creating a class without these fields consumers are not able to reach them.

5.2.2 API Documentation

To be able to consume the API an automatically generated documentation was generated by means of a package called Swagger. Swagger generates a web page for the API where all methods can be tested, along with documentation on how to use all the methods in the API, displaying requests in a JSON format.

5.2.3 Error Messages

To be able to tell the user if something went wrong with the HTTP request to any of the methods in the API exceptions with error codes and error messages are thrown and shown as a response from the server. This was accomplished by implementing a middleware that catches all exceptions thrown in the API project and prints out the code and message for each request to the user.

5.3 Skoruba

When the above implementations were completed an administration application became a priority. Skoruba was run as a separate application altering the current configuration and operational database of IdentityServer4 to include the Identity package. Skoruba can then be used to add administrators as well as new clients and scopes. The configuration within Skoruba also replaces the previous method of initially seeding the database with the new database configuration.

6
Result

The project aimed to supply QBIS with the same functionalities as previously featured in QBIS and improve upon them. In this section a comparison is drawn to those expectations and requirements with the final implementation provided in the test server. The IDP of choice was IdentityServer4 due to the factors of no cost for an applicable license, widespread use, familiar language/framework, excellent flexibility in features and good documentation availability.

6.1 Basic Authentication and Authorization

The IDP can be used for basic authentication and authorization as well as other ways which are listed below. By a user providing credentials, including company, username and password, the user can be authenticated and receive an id token. This server can be used by other products that QLogic might produce with some extra implementations in the framework that has been provided.

6.2 BankID

The IDP supports BankID authentication. This can be done on the same device, with a file, or on a separate device, where the certificate would be stored. The BankID authentication is counted as an external authentication service and serves as a basis for adding future external authentication services.

6.3 SAML Authentication

SAML authentication is not handled by the IDP. The current system will continue to handle the authentication needs of SAML users though it will eventually be provided by the new IDP as more external providers are included.

6.4 RESTful API

A protected API was implemented and follows the criteria to be considered RESTful; A GET, POST, PUT and DELETE method was considered enough implementation to expand upon in the future, giving the developer team all the tools needed to add

any additional support they wanted. The methods enable access to read or write to the database where all the employees are stored.

6.4.1 Retrieving Correct Data

To be able to retrieve the right data and limit access to all employees from all companies additional claims are added in the access token when a user logs in. When the access token is provided to the API each method in the API is able to connect to the right company database for the logged in user by reading from the claims in the token. The API connects to the company database for that specific user.

6.4.2 API Security

The API was configured to be protected by the IdentityServer instance, meaning to be able to access the API a valid access-token from the IdentityServer4 instance is required in each call to any of the API's methods.

6.4.3 API Documentation

In order for API consumers to be able to understand how to consume the API's different functionalities Swagger was used. Swagger provided a user interface where consumption of all methods of the API was available, shown in figure 6.1. Also descriptions of each method was explored in the Swagger user interface. In addition to this, Swagger generated a JSON-format where all API methods are provided with a description of how to use them.

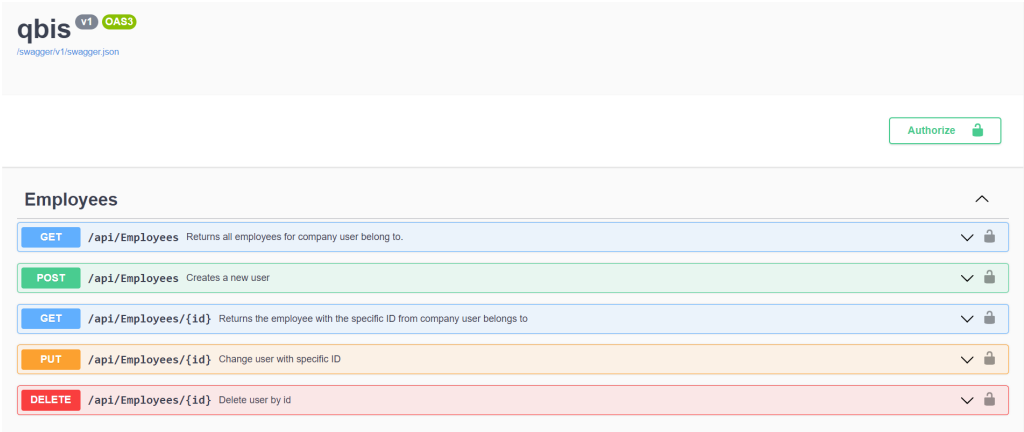

Figure 6.1: The user interface generated by Swagger.

6.5 Skoruba

An administration tool was connected to IdentityServer4 configuration stores. This required some minor adaptation of the configuration and operational store of IdentityServer4. This tool allows administrators to add, edit or remove clients that they

want connected to the back-end of the authorization server, without having to do so directly in code.

7
Discussion

7.1 The Choice of Identity Provider

While this project selected IdentityServer4 as the IDP of choice, there are plenty of other IDPs that would suit differing situations. Following is a reflection on factors affecting the choice of IDP in this project.

7.1.1 Maintenance

Maintainability can be tough to quantify. Arguments can be made that it is more maintainable by QBIS because they are happy with the framework it uses and that they can adapt it once it becomes an end-of-life project. It is difficult to predict what will happen with the large pool of IdentityServer4 users. They could move to a different product, keep their current implementations or fork IdentityServer4 and provide updates for it. One such project may become public and could be implemented for QBIS needs.

7.1.2 Usage and Documentation

Another aspect that was looked at was how widespread the usage of the software was. Particularly in the case of QBIS, where there is a complex database structure that is not cohesive with Identity and use of three parameters for input, it is incredibly helpful to find others that have tackled similar problems. This both ensures that these features can be implemented and that there are routes to explore when the desired functionality is not achieved.

7.1.3 Pricing

Pricing is clearly a huge factor. QBIS is hit quite hard by this as users will generally be logged in for a short time and need to authenticate again. The pricing schemes vary heavily from provider to provider. Some evaluate by clients and some do so by users. It can be an uncomfortable risk to have pricing vary based on how many clients one connects, which would potentially act as incentive to not offer a service to customers.

7.2 Further Developments

The project was made with expansion in mind. As such there are many areas where the project has been left in a state requiring additional implementation. Following are some of those areas.

An expansion of the REST API needs to be made. While implementations of POST and GET methods were made these are limited in that not all data can be fetched from the database and not all features for the employee information are functional either. Missing features include things such as hindering users without the correct scope to access the commands. These checks are easily added at a later time.

SAML login was not implemented in this project and will instead continue to use QBIS original implementation.

7.3 Ethical and Ecological Aspects

The project does not incur any negative ecological, societal or ethical consequences. The project could incur positive ethical effects by means of protecting sensitive client data and offering a valuable service to the company and its users.

7.4 The Effect of Agile

The project did not meet all the initial requirements that were set out. While the aforementioned sections are incomplete, the work method was agile and some change in scope was expected and did occur over the duration of the project. Earlier on, the scope shrunk significantly, but expanded towards the end of the project as various functions were implemented. Some functionalities that were not mentioned at the beginning of the project were introduced towards the end.

Overall all participants were pleased with the results of the project, since tasks had been accomplished in their highest priority over the duration of the project. Regular communication about the tasks allowed this.

7.5 Critical Discussion

There were issues with defining the proper boundaries of the project. In the early stages the project was laying the groundwork for a REST API and an IDP service. This then became researching the field for the best possible solution and then returning to the prior goal while adding on an administrator application. Part of this is working in an agile fashion, where a change in scope is required, and part of it was poor communication and knowledge regarding the size of the task at hand.

Researching took up a substantial amount of time early, since knowledge was lacking on most concepts within authentication and authorization as well as the working environment. This led to a slow start and it certainly would have been simpler to have a more defined boundary than finding the ideal IDP for QBIS.

The issue with this form of research and analysis is that one can not be sure that the answer is found. There may be another product that has better qualifications. The project likely should have settled on IdentityServer4 sooner than it did so that implementation could have started sooner. This would also have provided more answers regarding the weaknesses and strengths in the product.

Another issue was the vast amount of sections that needed to be implemented. The project consists of database configuration, database management, database abstraction, web development, security, authentication, front end and much more. This can be an overwhelming amount of fields for junior developers. Extra guidance was granted halfway through the project and significantly clarified priorities which needed to be implemented and this issue was partially solved.

8

Conclusion

This report demonstrates how an IDP could be implemented to act as an independent service of other services that a company might offer. The service is useful to the user as they get a familiar login screen for multiple services and the same forms of identification can be used for various applications. For the programmers it is beneficial because only a single front end is required for all the login necessities and authorization can be dealt with the same way in all endpoints. New changes to these systems will be applied throughout all the services and no longer need to be implemented individually for each new service.

The API will also be helpful as a stateless API, allowing an alternative mean of access to information that a client might need. This eases the development of other applications, both from a user and developer perspective, QBIS or not. Being able to access the data that QBIS holds allows users to create or configure their own applications based on it.

Finally the administration tool for user authentication will be centralised for all future products of QLogic within Skoruba. Within Skoruba client configurations can be set to appropriate values allowing altered behaviour from the IDP, suited to new products that QLogic release.

The project was not implemented in its entirety. Bugs remain to be fixed, features remain to be implemented and deployment still needs to be undertaken. The project has acted as more than simply a proof-of-concept and will be in use in the near future after further implementation. The report has covered many of the aspects one needs to understand to begin implementing an independent IDP such as authentication, authorization and some factors that are important to the choice of IDP such as licensing, pricing, popularity and documentation quality. There are strong benefits to having an independent IDP such as benefits to development and consistency in service.

Bibliography

[1] M. Augustsson, *High level requirements specification for master thesis*, 2022.

[2] T. Kodam, "A roadmap for ensuring saml authentication using identity server for on- premises and cloud," *Luleå Tekniska Universitet*, 2019. [Online]. Available: https://ltu.diva-portal.org/smash/get/diva2:1316547/FULLTEXT01.pdf (visited on 04/01/2022).

[3] G. Barbaglia, S. Murzilli, and S. Cudini, "Definition of rest web services with json schema," *Software: Practice and Experience*, vol. 47, no. 6, pp. 907–920, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1002/spe.2466 (visited on 05/18/2022).

[4] W. Li and C. J. Mitchell, "Analysing the security of google's implementation of openid connect," *Springer International Publishing*, 2016. [Online]. Available: https://www.researchgate.net/publication/303912803_Analysing_the_Security_of_Google's_Implementation_of_OpenID_Connect (visited on 04/01/2022).

[5] H. Z. U. Khan and H. Zahid, "Comparative study of authentication techniques," *International Journal of Video & Image Processing and Network Security IJVIPNS*, 2010. [Online]. Available: http://ijens.org/103304-2929%20IJVIPNS-IJENS.pdf (visited on 04/01/2022).

[6] H. S. Oluwatosin, "Client-server model," *IOSR Journal of Computer Engineering (IOSR-JCE)*, 2014. [Online]. Available: https://www.researchgate.net/profile/Shakirat-Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf (visited on 04/01/2022).

[7] H. Levy and M. Sidi, "Polling systems: Applications, modeling, and optimization," *IEEE Transactions on Communications*, 1990. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/61446 (visited on 04/01/2022).

[8] H. Seichter, R. Grasset, J. Looser, and M. Billinghurst, "Multitouch interaction for tangible user interfaces," *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5336455 (visited on 04/01/2022).

[9] T. S. I. Stanivuk V. Bjelić and Đ. Simić, "Expanding lua interface to support http/https protocol," *2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8246311 (visited on 04/01/2022).

[10] Microsoft, "The http status code in iis 7.0 and later versions," *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/troubleshoot/developer/webapps/iis/www-administration-management/http-status-code (visited on 04/01/2022).

[11] ——, "Om api management," *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/sv-se/azure/api-management/api-management-key-concepts (visited on 04/01/2022).

[12] ——, "Tutorial: Create a web api with asp.net core," *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio (visited on 04/01/2022).

[13] Identityserver4, "Welcome to identityserver4 (latest)," *Identityserver4*, 2022. [Online]. Available: https://identityserver4.readthedocs.io/en/latest/index.html (visited on 04/01/2022).

[14] Microsoft, "Soap web services," *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/webservices/soap-web-services (visited on 04/01/2022).

[15] ——, "Azure rest api reference," *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/rest/api/azure/ (visited on 04/01/2022).

[16] S. Safris, 2019. [Online]. Available: https://www.toptal.com/web/json-vs-xml-part-1 (visited on 06/07/2022).

[17] jwt, "Jwt," *jwt.io*, 2022. [Online]. Available: https://jwt.io/ (visited on 04/01/2022).

[18] OAuth, "What is oauth 2.0?" *auth0.com*, 2022. [Online]. Available: https://auth0.com/intro-to-iam/what-is-oauth-2/ (visited on 04/01/2022).

[19] Skatteverket, "Personal identity numbers and coordination numbers," *Skatteverket.se*, 2022. [Online]. Available: https://www.skatteverket.se/servicelankar/otherlanguages/inenglish/individualsandemployees/livinginsweden/personalidentitynumberandcoordinationnumber (visited on 04/01/2022).

[20] e-identitet, "E-identitet," *jwt.io*, 2022. [Online]. Available: https://e-identitet.se/auth/e-legitimation/id-vaxling/ (visited on 04/01/2022).

[21] O. Connect, "Openid connect," *openid.net*, 2022. [Online]. Available: https://openid.net/ (visited on 04/01/2022).

[22] Oracle, "Oracle," *Oracle.com*, 2022. [Online]. Available: https://www.oracle.com/se/database/what-is-database/ (visited on 04/01/2022).

[23] iso.org, "Iso," *iso.org*, 2016. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:9075:-1:ed-5:v1:en (visited on 04/01/2022).

[24] Okta, "Saml och oauth: Jämförelse och skillnader," *Okta.com*, 2022. [Online]. Available: https://www.okta.com/se/identity-101/saml-vs-oauth/ (visited on 04/01/2022).

[25] Shibboleth, "What is shibboleth?" *Shibboleth.net*, 2022. [Online]. Available: https://www.shibboleth.net/about-us/the-shibboleth-project/ (visited on 04/01/2022).

[26] Microsoft, "What is .net?" *Microsoft*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/shows/net-core-101/what-is-net (visited on 04/01/2022).

[27] J. Skoruba, "Skoruba user admin," *Github.com*, 2022. [Online]. Available: https://github.com/skoruba/IdentityServer4.Admin (visited on 04/01/2022).

[28] Atlassion, "Atlassian," *Atlassian.com*, 2022. [Online]. Available: https://www.atlassian.com/agile (visited on 04/01/2022).

[29] Microsoft, "Visual studio," *Microsoft.com*, 2022. [Online]. Available: https://visualstudio.microsoft.com/ (visited on 04/01/2022).

[30] Postman, "Postman," *Postman.com*, 2022. [Online]. Available: https://www.postman.com/ (visited on 04/01/2022).

[31] Swagger, "Swagger," *Swagger.io*, 2022. [Online]. Available: https://swagger.io/solutions/api-documentation/ (visited on 04/01/2022).

[32] GitHub, "Github," *GitHub.com*, 2022. [Online]. Available: https://github.com/ (visited on 04/01/2022).

[33] Synopsys, "5 types of software licenses you need to understand," *synopsys.com*, 2020. [Online]. Available: https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/ (visited on 04/01/2022).

[34] Keycloak, "Keycloak," *keycloak.org*, 2022. [Online]. Available: https://www.keycloak.org/ (visited on 04/01/2022).

[35] Microsoft, "Identityserver for asp.net core," *Microsoft.com*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/shows/introduction-to-identityserver-for-asp.net-core/identityserver-for-asp.net-core (visited on 04/01/2022).

[36] ——, "Introduction to identity on asp.net core," *Microsoft.com*, 2022. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio (visited on 04/01/2022).

[37] FusionAuth, "Fusionauth," *fusionauth.io*, 2022. [Online]. Available: https://fusionauth.io/ (visited on 04/01/2022).

[38] ActiveLogin, "What is active login?" *activelogin.net*, 2022. [Online]. Available: https://activelogin.net/ (visited on 04/01/2022).

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY